NPS CS-92-020

NAVAL POSTGRADUATE SCHOOL
Monterey, California

A Design Management and Job Assignment System

Salah M. Badr

Valdis Berzins

Prepared for:

Naval Postgraduate School
Monterey, California 93943

NAVAL POSTGRADUATE SCHOOL
Monterey, California

REAR ADMIRAL R. W. WEST JR..
Superintendent

HARRISON SHULL
Provost

This report was prepared for and funded by the Naval Postgraduate School.

This report was prepared by:

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): NPS CS-92-020	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Computer Science Dept. Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): CS	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Naval Postgraduate School; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

A Design Management and Job Assignment System

12. PERSONAL AUTHOR(S): SALAH M. BADR, VALDIS BERZINS'

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM 05/89 TO 03/90	December 1992	25

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	SOFTWARE EVOLUTION, JOB ASSIGNMENT, DESIGN DATABASE, EVOLUTION STEP, SOFTWARE PROTOTYPING.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report introduces the basic approach taken in designing the design management and job assignment system (DMJAS)for CAPS93. This approach uses a model of software evolution [8] which defines how a software change, once has been approved, will be applied to the software release (version) to produce another version of the software incorporating this change. This process is called an evolution step. The proposed system represents a management layer between the user interface (supporting two user classes, managers and designers) and the design database. The DMJAS controls the software evolution process in an incrementally evolving software system where The job steps to be scheduled are only partially known: time required, the set of sub-tasks for each step, and the input/output constraints between steps are all uncertain, and are all subject to change as steps are carried out. Models of design database and manager/designer interface are explained followed by the proposed system.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL: VALDIS BERZINS	22b. TELEPHONE (Include Area Code): (408) 656-2903	22c. OFFICE SYMBOL: CS/Lq

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE: UNCLASSIFIED

A Design Management and Job Assignment System[1]

Salah Badr
Valdis Berzins
Naval Postgraduate School
Department of Computer Science
Monterey, California 93943-5100 USA

ABSTRACT

This report introduces the basic approach taken in designing the design management and job assignment system (DMJAS) for CAPS93. This approach uses a model of software evolution [8] which defines how a software change, once has been approved, will be applied to the software release (version) to produce another version of the software incorporating this change. This process is called an evolution step. The proposed system represents a management layer between the user interface (supporting two user classes, managers and designers) and the design database. The DMJAS controls the software evolution process in an incrementally evolving software system where The job steps to be scheduled are only partially known: time required, the set of sub-tasks for each step, and the input/output constraints between steps are all uncertain, and are all subject to change as steps are carried out. Models of design database and manager/designer interface are explained followed by the proposed system.

KEYWORDS

SOFTWARE EVOLUTION, JOB ASSIGNMENT, DESIGN DATABASE,

EVOLUTION STEP, SOFTWARE PROTOTYPING,SOFTWARE ENGINEERING.

1. Introduction

"The Computer Aided Prototyping System (CAPS), an integrated set of computer aided software tools, has been

1. This research was supported in part by the Army Research Office under grant number ARO-145-91

designed to support prototyping of complex software systems. CAPS can increase the leverage of the prototyping strategy by reducing the effort of the designer puts into producing and adapting a prototype to perceived user needs." [7]

In the context of the CAPS system, there are three main entities that interact to develop a software system. These main entities are the project manager, the CAPS system, and the software designers. Currently CAPS92 has only two of these main entities: the designer and CAPS system. The designer can select a tool, then select a prototype to work with. The designer can commit his work to the design database whenever he wants. When a designer is modifying a prototype no other designer can access the same prototype for modification. Currently CAPS does not have any mechanisms for coordinating parallel efforts by different designers. This paper presents the design of such a mechanism for CAPS93.

The inter-relation between these three main entities in CAPS93 is depicted in figures 1, 2, and 3.The project manager should be able to enter/edit description of top level evolution steps. An evolution step has many properties and operations such as base version, change description, primary input, initial conditions, commit, abandon/suspend, and any optional data such as priority, or deadlines. The manager

should be able to monitor and control all the activities of the development team through making all the control information accessible to him such as the incremental task schedule describing who is supposed to do what and when. The Manager can act as a designer and can be assigned some development tasks.

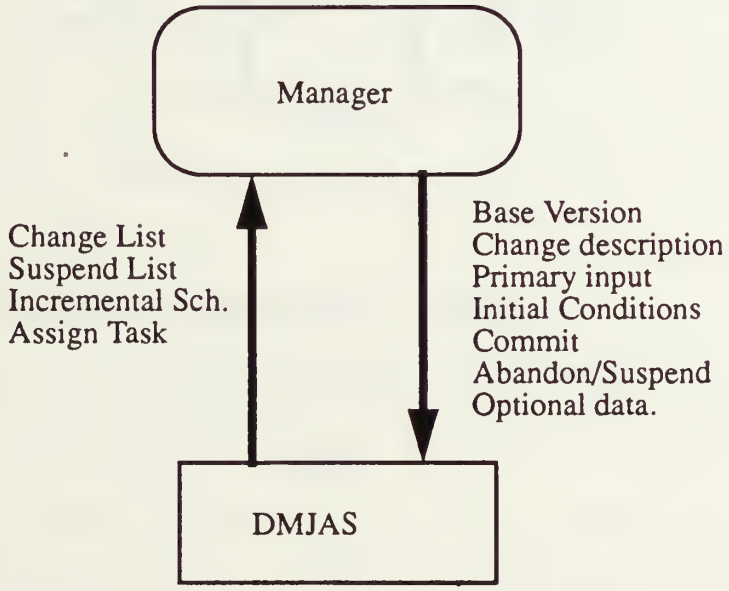

FIGURE 1. The DMJAS Manager Interface

The designer interface with the DMJA system is simplified to keep the designer's attention on doing the tasks assigned to him. This interface can assign tasks from the system to the designer and gets back his commit response. Designers can work on parallel on different tasks related to the same software

version, or on the same task splitting different variations (alternatives).

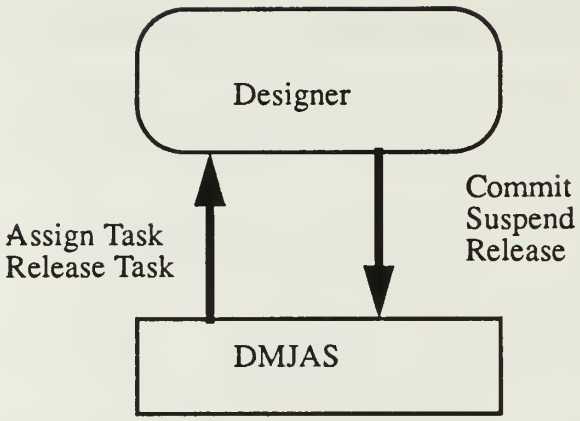

FIGURE 2. DMJAS Designer Interface

Version control and design management are also part of our contribution as tasks of the DMJA system as well as preserving data consistency through propagating change consequences. In section 2 some concepts used in the database schema for our work are defined. A comparison to the previous work is presented in section 3. In section 4 the underlying design database is defined. Sections 5 and 6 introduce both the designer and manager interfaces. The CMJA system is presented in section 7. Finally conclusions and suggestions for future work are presented in section 8.

2. Definitions

Since the DMJAS is a management layer on top of the design database, it gets most of its input data and sends most of its output data from/to the design database. It logs the evolution steps with their relevant information to the history log, assigns variation and version numbers to the new versions resulting from applying the evolution steps, queries the design database for the decomposition as well as the dependency relations between different objects, copies the tasks to be modified to the assigned designer's work area, and controls commitment of completed modifications to the database. The underlying database schema for our work includes the following concepts [8]:

An object is a software component that is subject to change. Objects can be either composite or atomic. Objects can be changed only by creating new versions. Object is a generalized type with many specialized subtypes that can include requirements, specifications, designs, code, test cases, etc.

A version is an immutable snapshot of an object. Versions have unique identifiers. New versions can be created, but versions cannot be modified after they are created.

A variation of an object is a totally ordered sequence of versions of the object which represent the evolution history of an independent line of development.

An **evolution step** represents the activity of creating a new version of the source description of a software object. Evolution steps usually require creativity and human effort from the manager responsible for the step.

A software component or step is **composite** if it can be viewed as a collection of related parts, and is **atomic** otherwise.

A **top level Evolution Step** represents the activity of initiating, analyzing, and implementing a change request in the system.

X uses Y: a relationship that is true if and only if the semantics or implementation of one software object X depends on another software object Y [4].

X used-by Y: same as Y uses X.

3. Previous Work

According to [11] representations of the versioning process can be classified into two main models. The first model is the conventional Version Oriented Model VOM in which a

system is divided into modules each of which is versioned independently from the other modules. To configure a system one has to select a version of each module of the system. This makes version a primary concept while change is a secondary concept as a difference between versions. Both SCCS and RCS belongs to this model.

The second model is the Change Oriented Model COM. In this model the functional change is the primary concept. Versions are identified by a characteristic set of functional changes. To configure a system in this model one has to select a set of mutually compatible functional changes. Versions in this model are global, meaning that to examine a module one has to specify a single version of the system first, then proceed to the required module. On the other hand, in a VOM system, to examine a module one has to select the module first then individually select which version of this module is the target. Our work utilizes concepts from both models. A set of a changes to a base version of a software system leads to the versioning of both the individual objects involved in the changes and also acts on the entire software system producing a new release (version of the whole system).

According to Kaiser and Perry [12] the main tools that propagate changes among modules are the following. However, none of these support the enforced model of cooperation among programmers necessary for large maintenance/evolution projects or automatically assign tasks to programmers:

Make: a UNIX tool that rebuilds the entire software system. It invokes the tools specified in the 'makefile' on changed files and their dependent files. Make is used for regenerating up to date executables after source objects have been changed.

Build: is an extension to make that permit various users to have different views of target software system. A 'viewpath' defines a series of directories to be searched by make to locate the files listed in the makefile.

Cedar: the Cedar System Modeler uses an advanced version of the Make tool with version control to invoke the tools on a specific versions of files. This System informs the 'Release Master', a programmer, about any syntactic interface errors. The Release Master is responsible for making work arrangements with responsible programmers.

DSEE: the Apollo Domain Software Engineering Environment also uses a Make-like tool with version control. DSEE also has a monitoring facility that permits programmers and/or managers to request to be notified when certain modules are changed.

Masterscope: Interlisp's Masterscope tool maintains cross-reference information between program units automatically. It also approximates change analysis of potential interference between changes by answering queries about syntactic dependencies among program units.

SVCE: the Gandalf System Version Control Environment performs incremental consistency checking across the modules in its database and notifies the programmer of errors as soon as they occur. The consistency checking is limited to syntactic interface errors. It supports multiple programmers working in sequence but does not handle simultaneous changes.

Kaiser and Perry [12] also describe *Infuse*, a system that automates change management by enforcing programmers cooperation to maintain consistency among a sequence of scheduled source code changes. Infuse automatically partitions these modules into a hierarchy of experimental databases. This partitioning may be done according to the syntactic and/or

semantic dependencies among the modules or according to project management decision. Each experimental database provides a forum for the programmers assigned to its modules or their managers, and provides also for consistency checking among those modules (meaning that the interface between the modules must be correct and that the modules can compile and link successfully). Consistency checking among the experimental database modules is a pre-condition for merging a database back to its parent experimental database. Infuse automatically partitions the database into experimental databases but programmers are assigned to the these databases manually. In our system tasks are assigned directly to designers (programmers) according to their (uses) dependencies. Versions are generated automatically as soon as the work is done. Syntactic and semantic consistency checking for source code can be implemented by associating declarations of consistency constraints with steps, and triggering the required checking actions as part of the commit protocol.

4. Design Database

The main goal of the CMJA system is to provide automated design data support for software development through all phases of the software life-cycle and to provide bilateral

communication between designers/programmers [7] as well as providing for the management control over all the ongoing activities. Keeping this objective in mind besides complying with both the Graph Model Of Software Evolution [8] and the ANSI/IEEE standard [5], the design data base should have the following characteristics [7]:

- Minimize the communication time between designers/programmers through keeping the development documents on-line.
- Propagate change consequences to maintain the global consistency of the database.
- Produce nondisruptive status reports for an ongoing project.
- Provide a way of tracking the development history via keeping an up-to-date history file on-line that has all the design decisions and all relevant information to development history.
- Support planning and scheduling any proposed changes/maintenance by keeping the relationship between different data objects.
- Support software reusability to save both cost and effort.

The proposed system will take care of the following function:

- Propagate change consequences.
- Schedule evolution and maintenance tasks to maximize concurrency and minimize rollbacks.
- Record development history.
- Provide version and configuration management.

The interface between the DMJAS and the design database is illustrated in figure 3.

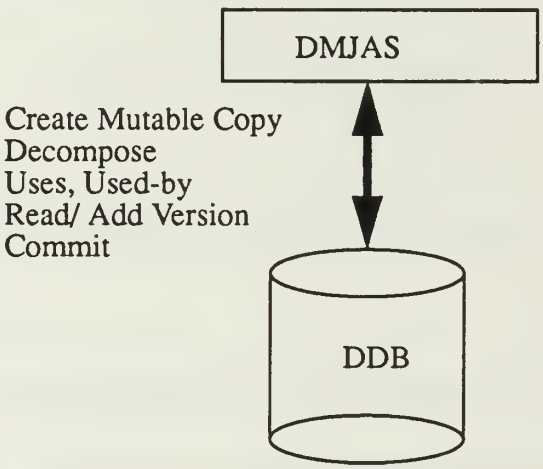

FIGURE 3. The DMJAS Design Database Interface

The design database is divided into two main parts:

- Shared data space where the frozen software objects are stored, and
- Private workspaces where the development of new software objects/versions takes place.

4.1 Shared data space

The shared data space is the repository that keeps all of the verified software objects (versions or configurations) [5]. The versions in the shared data space are frozen and may not be changed under any circumstances. Any changes to any of the objects must be authorized by the management and this will produce new versions. The shared data space contains the public releases of the software objects and these objects may be checked out in write-protect mode only. The relations between the objects in the database are kept in a form accessible to the design management and job assignment system (DMJAS).

4.2 Private workspaces

Since the data in the shared data space is frozen and may not be changed, the private workspaces are used for production of new versions of existing objects or adding new objects to existing software systems.

The mutable data contains a copies of the specific versions of the software to which the changes are to be implemented (base versions). Only the designer responsible for an evolution step has access to the mutable copy of the base version in that designer's private workspace. The process of copying objects to and from the designer's workspace is done

automatically by DMJAS, and these objects continue to be under its control until either all the changes are done and the DMJAS commits them to generate the new version. When a mutable version in the design database is committed by the DMJAS it becomes a immutable version in the shared data space. If the changes are suspended/abandoned then the mutable versions are appended to the abandoned or suspended log respectively.

5. Designer Interface

The designer interface with the proposed system is meant to be very simple, where the system assigns tasks to each designer according to the initial data it has about the design team and the constraints given by the project manager. The DMJAS copies those tasks automatically to the designers working areas. As soon as the designer finishes his task he can commit it by issuing the commit command upon which the DMJAS moves the task back to its internal database. The DMJAS keeps track of the status of the sub-steps of a top level evolution step and releases a new version to public view only when all of the sub-steps have been committed. The DMJAS can then assign new task to the designer (if it has any). The designer can also suspend a task for some time then release it back when he is ready. The DMJAS adds the sus-

pended task to the suspend log, and does not assign the designer any more tasks until he releases the suspended one and commits it.(Such cases may happen if a designer takes a short leave and his work may not affect the rest of the team, otherwise the manager should appoint someone else to do the job).

A typical scenario illustrating the operation of the designer interface follows. A designer D1 receives a message from the system that he has been assigned a task T1 with information about the parent step, whether these tasks are primary or induced, the specification of the changes required, and the time of the assignment. When D1 is ready to commit his task he would invoke the system and commit the task which may lead to another assignment if there is any. The designer can also receive messages from the DMJAS directing him to abandon a task or to suspend activity on a task and switch to some other task. Such messages are responses to policy decisions made by the project manager.

6. Manager Interface

The manager interface is a super-set of the designer interface. Change requests are analyzed and entered into the DMJAS through the analyst's interface, which is beyond the

scope of this paper. The analysis of a change request determines a precise description of the changes required as well as the primary inputs of the objects to be changed are defined. The manager approves a change request, reviews the primary inputs, and the change description, and may specify optional information about a step regarding its priority relative to other steps, preference about who should do a step, and deadlines. The manager also has to decide up front if committing the changes has to pass any automated checking procedures or any manual review procedures that are tracked by the DMJAS. The DMJAS responds with:

- A list of the objects that have to be changed by decomposing the primary input(s) (if any is composite) and listing those objects that use any of the primary objects (steps to update these objects are called induced changes).
- A list of the objects in lower-priority ongoing changes that have to be suspended because of a dependency relation on the objects of this evolution step.
- An incremental job schedule derived from the number of jobs to be assigned, the available designers, and their current work loads, taking into consideration any currently suspended jobs. The manager can also add new sub-steps to an ongoing step, and suspend or abandon the whole step or its

sub-steps for any management reasons. As a designer the manager can be assigned an object, edit it and commit it. Figure 1. shows the DMJAS Manager interface.

7. A Design Management and Job Assignment System (DMJAS).

The design management and job assignment system represents a management layer between the interfaces to managers, analysts, and designers, and to the design database. The DMJAS controls the software evolution process in an incrementally evolving software system with the following special characteristics:

a. The job steps to be scheduled are only partially known: time required, the set of sub-tasks for each step, and the input/output constraints between steps are all uncertain, and are all subject to change as steps are carried out.

b. The method must support incremental replanning as additional information becomes known, and it must minimize lost work due to reorganization of the schedule as well as workers forced to wait for completion of sub-tasks in this uncertain environment.

c. Rescheduling of some jobs is required if new dependency constraints arise during scheduling of new jobs which needs the completion of the newly scheduled job before already scheduled (in-progress) jobs. This will lead to the suspension of the dependent jobs and rescheduling them accordingly. This function must be integrated with the design database access protocol because rescheduling can imply restarting an ongoing job step on different versions of its input objects (documents).

d. Allocation of agents (designers) can vary to fulfill the deadline constraints, and in response to changing estimates of the effort required for each step.

e. For those jobs that do not have a deadline the system should provide for earliest possible completion subject to the other constraints.

f. The goal is to determine and carry out a feasible schedule that meets the deadline requirements, provides for maximum concurrency, reduces/avoids rollbacks, and insures system integrity, while minimizing the amount of details that must be explicitly considered by the project manager.

The functions of the proposed system are concerned with three main tasks: version control. job scheduling and job assignment. This system is intended to support a prototyping

environment where frequent changes to evolving system are requested/proposed, evolution steps are triggered after analyzing the change requests, primary inputs are decomposed, induced changes are inferred, then jobs are incrementally assigned to the designers. When a designer finishes his job, it is committed producing new version of the committed object. When an evolution step is committed it produces a new version on a specific variation line. The following are the DMJAS detailed functions:

- Get and document the change request information in the software history file.
- Prepare the mutable copy of the defined software version for incorporating the required changes by a given evolution step and assign a new version identifier to it according to a well defined labeling function. This labeling function has the following restrictions:

  a. The ordering of the version identifiers must be the same as the serialization order of the changes that produced the version on a variation line.

  b. Changes to versions that are not the most recent on their variation line split off a new variation line and produce the first version on the new variation.

- Get the primary inputs of the evolution step then

a. decompose them if they are composite by querying the database for the image in the decomposition relation.and

b. find the induced changes by querying the database for the used-by relation for each input object resulted from step a above.

c. repeat step a, b until no further objects are found.

d. check if any of the above objects is "used-by" any of the objects of the ongoing changes (by any of the previous evolution steps that is not committed yet). If any is found either issue a warning to the manager or suspend the corresponding step according the policies specified by management (auto-suspend or warning).

- Build the dependency acyclic graph of the resulting objects from step 3.
- Build the lists of the changes that can be done concurrently and the dependencies between these changes. Display a copy to the manager [as a dependency graph].
- Assign the objects to the designers according to the following:

a. The author of each object.

b. Names of the designers, and

c. The status of the designer [free or busy].

d. If list X2 depends on list X1 start by assigning the objects of list X1, after each object in X1 is committed, check if an object in X2 can be assigned, if so assign it, as soon as the objects of list X1 are com-

mitted, assign the remaining objects of list X2, and so on.

- Commit the step: as soon as all the objects of a top level evolution step are committed and according to the specified management policies either:

  a. commit the step generating a new version of the software, or

  b. Notify the manager that all the modification are done and ask for permission to commit.

- Respond to the manager commands commit, suspend, abandon with the corresponding action.

  a. commit: commit the step generating a new version by changing the mutable data to shared data space and update the history file by the date and time the version is committed.

  b. Suspend: by copying the modified data to the suspended log with the reason why? and keep the version labeling.

  c. Abandon: by copying the modified data to the abandoned log with the reason why? and release the version labeling.

  d. Respond to the designer commands with the corresponding action: a. "Suspend" by copying the suspended object to suspend log. b. "Commit" by copying the object back to the mutable database and check it done. c. "Release" by releasing the suspended object if there is one and copying it back from the suspended log to the designer's work area.

7.1 Management Issues

The DMJAS partially automates most of the management functions whether they are design data management or human

management (design team). The proposed system automatically creates mutable software versions in the private workspaces from the immutable base version, automatically constructs breakdowns of evolution steps to their atomic components both primary and induced, incrementally assigns tasks to designers and keeps track of which jobs are assigned to which designer, when assigned and when committed. This information is available to the software manager who can judge the status of the project and the productivity of the team. The system also automatically produces the new versions or splits off new variations depending on the base version.

7.2 Job Scheduling

Job scheduling is completely automated since the system builds the dependency graph, prepare the dependency lists and assign a task to each designer based on his status, and his name. A task is chosen for a designer if he is not busy and the priority goes to the task with his name as the author if there is one, otherwise to the task with the maximum "used-by" relations (to release as many dependency conflicts as possible). As soon as a designer is done with a task by committing it he will be assigned another task on the same basis. The committed task is checked done, its commit time is recorded and its dependency relations are released.

The suspension of jobs because of dependency conflicts is done automatically and those jobs are copied to the suspended log, the corresponding designers are assigned new jobs. These suspended jobs are placed in the corresponding dependency list to be assigned later. When a suspended job is assigned again it is copied from the suspended log back to the assigned designer's work area, and the validity of the saved work is determined based on whether the input documents were bound to new versions since the time when the job was suspended and the version bindings for its inputs were released.

8. Conclusion

Our design management and job assignment system is a design management layer between the manager/designer interface and the design database that contains all the project software objects. The goal of this system is to help the project management to efficiently direct the efforts of the design team and to assure data consistency by propagating the change consequences to all the affected objects, automating the identification of implied sub-steps, automating job assignment and keeping track of the activity of each designer so the project manager can watch the progress of the team closely and determine when corrective actions become

necessary. Version control and design management are also automated. Creation of new versions on the same variation line with a serializable effect or splitting a new variation to do parallel updates without locking are done under the control of the system. Work on automated support for bringing a split variation lines back together is in progress.[13] [14].

LIST OF REFERENCES

[1] Berzins and Luqi, "Software Engineering with Abstractions", Addison-Wesley 1990

[2] Borison E., "A Model of Software Manufacture", in Advanced Programming Environment, Springer-Verlag, 1986, pp. 197-220.

[3] Feldman S. I., "Software Configuration management: Past Uses and Future Challenges" Proceedings of 3rd European Software Engineering Conference, ESEC '91, Milan, Italy, October 1991

[4] Heimbigner D. and Krane S., "A graph Transform Model for Configuration Management Environments", Proceedings of the ACM SIGSOFT/ SIGPLAN, Nov. 28-30, 1988.

[5] "IEEE Guide to Software Configuration Management", Std 1042-1987, American National Standards Institute/IEEE, New York, 1988.

[6] Ketabchi M. A., "On the Management of Computer Aided Design Database", Ph. D. Dissertation, University of Minnesota, Nov. 1985.

[7] Luqi, "Software Evolution Through Rapid Prototyping", IEEE Computer 22, 5. May 1989, pp 13-25.

[8] Luqi, "A Graph Model for Software Evolution", IEEE Transaction on Software Engineering. Vol. 16. NO. 8. Aug. 1990

[9] Mostov I., Luqi, and Hefner K., "A Graph Model for Software Maintenance", Tech. Rep. NPS52-90-014, Computer Science Department, Naval Postgraduate School, Aug. 1989.

[10] Narayanaswamy K. and Scacchi W., "Maintaining Configuration of Evolving Software System", IEEE Trans. on Software Eng. SE-13,3. Mar. 1987, pp. 324-334.

[11] Lie A. et al, "Change Oriented Versioning in a Software Engineering Database", Proceedings of 2nd International Workshop on Software Configuration Management, Princeton, New Jersey, Oct. 24,1989. pp. 56-65.

[12] Kaiser G. E., and Perry D. E., "Workspaces and Experimental Databases: Automated Support for Software Maintenance and Evolution", Proceedings of IEEE conference on Software Maintenance 1987. pp. 108-114.

[13] D. Dampier, "A Model for Merging Different Versions of a PSDL Program", MS thesis, Computer Science, Naval Postgraduate School, June 1990.

[14] D. Dampier, Luqi, "A Model for Merging Software Prototypes", Technical Report, CS, NPS, 1992.

DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Director of Research Administration, Code 08 Naval Postgraduate School Monterey, CA 93943	1
Library, Code 52 Naval Postgraduate School Monterey, CA 93943	2
Egyptian Military Attache 2308 Tracy Place NW Washington, DC 20008	2
Egyptian Armament Authority - Training Department c/o American Embassy (Cairo, Egypt) Office of Military Cooperation Box 29 (TNG) FPO, NY 09527-0051	2
Military Technical College (Egypt) c/o American Embassy (Cairo, Egypt) Office of Military Cooperation Box 29 (TNG) FPO, NY 09527-0051	5
Military Research Center (Egypt) c/o American Embassy (Cairo, Egypt) Office of Military Cooperation Box 29 (TNG) FPO, NY 09527-0051	5
LTC. Salah M. Badr Computer Science Department, Code CS Naval Postgraduate School Monterey, CA 93943	10
Dr. Luqi Computer Science Department, Code CS Naval Postgraduate School Monterey, CA 93943	1

Dr. Valdis Berzins 10
Computer Science Department, Code CS
Naval Postgraduate School
Monterey, CA 93943

Dr. Mantak Shing 1
Computer Science Department, Code CS
Naval Postgraduate School
Monterey, CA 93943

Michael L. Nelson, Maj. USAF 1
HQ USCINCPAC / J66
Box 32A
Camp H. M. Smith, HI 96861

Dr.Tarek Abdel-Hamid 1
Administrative Science Department, Code AS
Naval Postgraduate School
Monterey, CA 93943

Dr. J. T. Butler 1
Electrical and Computer Engineering Department, Code EC
Naval Postgraduate School
Monterey, CA 93943